МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

О.І. Марченко, О.О. Марченко

СТРУКТУРИ ДАНИХ ТА АЛГОРИТМИ

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СОРТУВАННЯ НА БАГАТОВИМІРНИХ МАСИВАХ

КУРСОВА РОБОТА

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 123 «Комп'ютерна інженерія»

Київ

КПІ ім. Ігоря Сікорського

2018

Рецензент: *Заболотня Т.М., канд. техн. наук, доцент*
Відповідальний редактор: *Орлова М.М., канд. техн. наук, доцент*

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол №10 від 21.06.2018 р.) за поданням Вченої ради ФПМ (протокол №11 від 04.06.2018 р.)

Електронне мережне навчальне видання

Марченко Олександр Іванович, канд. техн. наук, доцент
Марченко Олексій Олександрович, асистент

СТРУКТУРИ ДАНИХ ТА АЛГОРИТМИ
ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ СОРТУВАННЯ НА БАГАТОВИМІРНИХ МАСИВАХ
КУРСОВА РОБОТА

Структури даних та алгоритми: Дослідження ефективності алгоритмів сортування на багатовимірних масивах: Курсова робота [Електронний ресурс] : навч. посіб. для студ. спеціальності 123 «Комп'ютерна інженерія» / О.І. Марченко, О.О. Марченко ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 0,7 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 73 с.

Навчальний посібник розроблено для ознайомлення студентів із завданнями курсової роботи по кредитному модулю «Структури даних та алгоритми. Курсова робота». Навчальний посібник містить інформацію до виконання проекту програми курсової роботи, яка включає постановку завдання та вимоги до проекту програми, що повинні розробити студенти, вказівки до вимірювання часу роботи алгоритмів, вказівки до оформлення звіту та тестування розробленого алгоритму і відповідної йому програми, варіанти індивідуальних завдань. Навчальний посібник призначений для студентів очної форми навчання за спеціальністю 123 – «Комп'ютерна інженерія» факультету прикладної математики КПІ ім. Ігоря Сікорського.

ЗМІСТ

ЗМІСТ.....2

ВСТУП.....4

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ 6

ВИМОГИ ДО ПРОГРАМИ КУРСОВОЇ РОБОТИ.....9

СТРУКТУРА ЗВІТУ КУРСОВОЇ РОБОТИ.....10

ОЦІНЮВАННЯ КУРСОВОЇ РОБОТИ 12

ІНСТРУКЦІЇ З ВИМІРУ ЧАСУ РОБОТИ АЛГОРИТМІВ 15

ЗАДАЧІ 23

АЛГОРИТМИ СОРТУВАННЯ 24

СПОСОБИ ОБХОДУ (ВЗЯТТЯ ЕЛЕМЕНТІВ) МАСИВА ДЛЯ ВИКОНАННЯ СОРТУВАННЯ 26

ВИПАДКИ ВІДСОРТОВАНОСТІ МАСИВА 28

ВИПАДКИ ДОСЛІДЖЕННЯ.....29

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ 37

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ.....42

ДОДАТОК 1. АЛГОРИТМИ ДЛЯ ВИКОНАННЯ КУРСОВОЇ РОБОТИ 43

ДОДАТОК 2. ПЕРШИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ. 68

ДОДАТОК 3. ДРУГИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ. 69

ДОДАТОК 4. СТРУКТУРА ТЕХНІЧНОГО ЗАВДАННЯ КУРСОВОЇ РОБОТИ. 70

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ 70

ВСТУП

Дисципліна "Структури даних та алгоритми" є базовою спеціальною нормативною дисципліною підготовки фахівців рівня бакалавр спеціальності 123 «Комп'ютерна інженерія», спеціалізацій «Комп'ютерні системи та компоненти», «Системне програмування» та «Спеціалізовані комп'ютерні системи», і викладається у 1-му та 2-му семестрах.

Вона повинна передувати всім програмістським дисциплінам крім початкового курсу програмування, з яким викладається одночасно.

Кредитний модуль «Структури даних та алгоритми. Курсова робота» для фундаментального засвоєння теоретичних знань і практичних навичок, отриманих під час вивчення дисципліни «Структури даних та алгоритми».

Курсова робота складається з практичної частини, яка полягає у реалізації заданих алгоритмів для рішення задач на багатовимірних масивах з використанням принципів структурного та модульного програмування, а також з теоретичної дослідницької частини, яка полягає у вимірюванні часу роботи алгоритмів як для звичайних випадків одновимірного масиву, так і різних випадків поведінки алгоритмів на багатовимірних масивах, і у дослідженні причин неоднакової поведінки цих алгоритмів на одновимірних і багатовимірних масивах.

Курсова робота дозволяє отримати досвід реалізації і дослідження складних алгоритмів на складних структурах даних, а також практичного засвоєння принципів структурного і модульного програмування і оформлення документації на створений програмний продукт.

Інструкції включають:

- постановку завдання та вимоги до виконання програми, що повинні розробити студенти;
- інструкції до вимірювання часу роботи алгоритмів;
- вказівки до оформлення звіту та тестування розробленого алгоритму і відповідної йому програми;
- варіанти індивідуальних завдань.

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

I. Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масива, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

II. Скласти алгоритми рішення задачі сортування в багатовимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

- Одна таблиця результатів (вимірів часу сортування впорядкованого, випадкового і обернено-впорядкованого масива) для масива з заданими геометричними розмірами повинна бути такою:

Таблиця № для масива A[P,M,N], де P= ; M= ; N= ;

	Впорядкований	Невпорядкований	Обернено впорядкований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

УВАГА! Колонки таблиць виміру часу в програмі та у звіті курсової роботи місцями НЕ переставляти! За невиконання цієї вимоги нараховується 3 (три) штрафних бали.

- Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.
- Для виконання ґрунтовного аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.
- Зробити виміри часу для стандартного випадку одномірного масива, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масива.
- Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.
- Рекомендації випадків дослідження з різними геометричними розмірами масивів наведені у розділі «Випадки дослідження».

VI. Для наочності подання інформації за отриманими результатами рекомендується також будувати стовпчикові діаграми та графіки.

VII. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами (вимірами часу):

- для одномірного масива відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одномірного масива;
- для заданих алгоритмів на багатовимірних масивах між собою;
- дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
- **для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше**.

УВАГА! Саме якість виконання порівняльного аналізу дозволяє отримати оцінки від B (85 балів) до A (100 балів). Якщо в звіті в якості порівняльного аналізу наведений просто словесний опис фактів, що й так видно з таблиць, діаграм та графіків, то оцінка курсової роботи буде не вище, ніж C (75 балів).

VIII. Зробити висновки за зробленим порівняльним аналізом.

IX. Програму курсової роботи під час її захисту ОБОВ'ЯЗКОВО мати при собі на електронному носії інформації.

ВИМОГИ ДО ПРОГРАМИ КУРСОВОЇ РОБОТИ

1. Всі алгоритми повинні бути реалізовані в рамках ОДНІЄЇ програми з діалоговим інтерфейсом для вибору варіантів тестування та виміру часу кожного алгоритму.

2. Одним з варіантів меню діалогового інтерфейсу програми має бути режим запуску виміру часу всіх алгоритмів у пакетному режимі, тобто запуск всіх алгоритмів для всіх випадків і побудова результуючої таблиці за наведеним вище зразком для масива з заданими геометричними розмірами.

3. Після декомпозиції задачі курсової роботи на підзадачі, для реалізації підзадач верхнього рівня повинні бути використані модулі (для мов C/C++ – це пара файлів “name.h” та “name.c”) із дотриманням виконання принципа «приховування інформації», як було викладено на лекціях.

4. Програма повинна мати коментарі для всіх структур даних, процедур та функцій, а також до основних смислових фрагментів алгоритмів.

5. Виміри часу роботи алгоритмів у програмі курсової роботи повинні строго задовольняти вимогам, зазначеним у розділі «Інструкції з виміру часу роботи алгоритмів».

СТРУКТУРА ЗВІТУ КУРСОВОЇ РОБОТИ

1. Перший титульний аркуш (додаток 2).
2. Другий титульний аркуш (додаток 3).
3. Технічне завдання (ТЗ) на курсову роботу. Загальна частина та структура індивідуальної частини ТЗ гаведені у додатку 4.
4. Опис теоретичних положень.
5. Схема імпорту/експорту модулів програми курсової роботи.
6. Структурна схема взаємовикликів процедур та функцій програми курсової роботи.
7. Опис призначення всіх функцій і процедур та їх параметрів.
8. Текст програми з коментарями.
9. Тести, що демонструють коректну роботу кожного із заданих алгоритмів для кожного випадку початкової впорядкованості масива.
10. Результати (таблиці виміру часу та побудовані за даними таблицями стовпчикові діаграми та графіки).
11. Порівняльний аналіз алгоритмів **(це ключовий пункт для отримання високих оцінок з діапазону 85 – 100 балів!!!)**
12. Висновки по отриманих результатах.
13. Список літератури.
14. Сторінки звіту пронумерувати починаючи з другої.

Листки звіту повинні бути скріплені довільним способом так, щоб звіт можна було читати як книжку і він не розпадався.

Звіт курсової роботи у вигляді файлу формату текстового редактора *MS WORD або Open Office Writer*, а також проект програми курсової роботи на мові програмування разом з файлами даних повинні бути:

1) зархівовані у архів з ідентифікатором виду **KV–XX_PRIZVYSCHE** (ідентифікатор записувати **ТІЛЬКИ ЛАТИНСЬКИМИ** буквами);

2) вислані електронною поштою на адресу **sda_kr@ukr.net**.

ОЦІНЮВАННЯ КУРСОВОЇ РОБОТИ

Семестровий рейтинг з курсової роботи «Дослідження алгоритмів» дисципліни «Структури даних та алгоритми» визначається згідно таблиці 1 і не може перевищувати 100 балів. Якщо студент з урахуванням заохочувальних балів отримує більше 100 балів, йому проставляється 100 балів.

Таблиця 1. Система рейтингових балів.

№	Контрольний захід	Бали
	ПЕРЕВІРКА ЗВІТУ	
1	Загальний вигляд та правильність структури звіту.	2
2	Правильність оформлення титульних листів.	3
3	Правильність оформлення технічного завдання.	3
4	Якість викладення теоретичних положень.	5
5	Якість схеми взаємозв'язків модулів та схеми взаємовикликів процедур і функцій програми курсової роботи.	4
6	Якість опису призначення процедур і функцій.	4
7	Використання модулів у програмах.	5
8	Всі алгоритми реалізовані правильно та однаково ефективно.	5
9	Коректність реалізації виміру швидкодії алгоритмів.	5
10	Якість коментарів у програмі.	3
11	Наявність достатньої кількості тестів.	3
12	Наявність достатньої кількості таблиць результатів виміру часу.	5
13	Правильність вибору розмірів структур даних (масивів) для отримання результатів, які дозволяють коректне порівняння результатів.	5
14	Наявність вимірів швидкодії та порівняння з вектором.	5
15	Наявність порівняльного аналізу алгоритмів на багатовимірних масивах.	5
16	Повнота, коректність та конкретність порівняльного аналізу результатів у висновках.	16
17	Наявність переліку використаної літератури.	2
	Разом за звіт	**80**

	ЗАХИСТ	
18	Ступінь володіння матеріалом.	10
19	Аргументованість відповідей.	10
	РАЗОМ	**100**

Останній день здавання звітів курсової роботи на перевірку з максимально можливою оцінкою (100 балів) після захисту, встановлюється на останньому або передостанньому тижні квітня.

Якщо звіт курсової роботи був зданий несвоєчасно, застосовуються **штрафні санкції** за схемою показаною у Таблиці 2. Інші штрафні санкції перелічені в Таблиці 3.

Таблиця 2. Штрафні санкції за несвоєчасно зданий звіт.

№	Термін здавання звіту КР	Штрафна санкція
1	Курсова робота здана не пізніше дня лекції на другому тижні травня	Студент отримує оцінку не вище B, незалежно від набраних балів.
2	Курсова робота здана не пізніше дня лекції на третьому тижні травня	Студент отримує оцінку не вище C, незалежно від набраних балів.
3	Курсова робота здана не пізніше дня лекції за 3 тижні до кінця семестру.	Студент отримує оцінку не вище D, незалежно від набраних балів.
4	Курсова робота здана пізніше ніж в день лекції за 3 тижні до кінця семестру.	Студент отримує оцінку не вище E, незалежно від набраних балів.

Таблиця 3. Інші штрафні санкції.

№	Штрафна ситуація	Штрафна санкція
1	Старт і стоп таймера виконані в неправильних місцях програми (див. розділ «Інструкції з виміру часу роботи алгоритмів»).	6 (шість) штрафних балів
2	Виміри часу з тактів переведені у секунди (час поділений на CLOCKS_PER_SEC).	6 (шість) штрафних балів
3	Колонки в таблицях виміру часу розташовані інакше, ніж у заданому в технічному завданні зразку.	3 (три) штрафних бали

Заохочувальні бали можуть бути нараховані за дострокове здавання курсової роботи не менше ніж на тиждень до встановленої дати здавання на максимальну оцінку (5 балів); за глибину та особливу докладність порівняльного аналізу (від 2-х до 10-ти балів); за виконання курсової роботи за індивідуальним завданням підвищеної складності (від 10 до 20 балів).

УВАГА! Заохочувальні бали за не нараховуються, якщо виявлена несамостійність виконання курсової роботи.

У випадку виявлення несамостійності виконання курсової роботи бали за захист курсової роботи студенту не нараховуються, а також нараховується до 20 штрафних балів в залежності від рівня несамостійності або відразу проставляється оцінка «незадовільно».

Студент допускається до захисту при виконанні двох умов:

1) якщо його **рейтинг за звіт** з урахуванням штрафних та заохочувальних балів складає не нижче, ніж 40 балів, та

2) якщо **семестровий рейтинг** з СДА на день захисту не нижче, ніж 40 балів.

Таблиця 4. Відповідність між рейтингом та оцінкою .

Рейтинг	Оцінка ECTS	Традиційна оцінка
95≤RD≤100	A	Відмінно
85≤RD≤94	B	Добре
75≤RD≤84	C	Дуже добре
65≤RD≤74	D	Задовільно
60≤RD≤64	E	Достатньо
40≤RD≤59	FX	Незадовільно
0≤RD≤39	F	Не допущено

ІНСТРУКЦІЇ
З ВИМІРУ ЧАСУ РОБОТИ АЛГОРИТМІВ

УВАГА!

ВАЖЛИВІ ВИМОГИ ДО ВИМІРУ ЧАСУ:

1. Старт і стоп таймера повинні бути виконані прямо в тексті функції сортування безпосередньо перед першим оператором алгоритму сортування (старт) і безпосередньо після останнього оператора алгоритма сортування (стоп), як показано нижче у прикладі функції ExchangeSort. Виключенням є тільки рекурсивний алгоритм методу Швидкого сортування (Хоара), для якого старт таймера повинен бути безпосередньо перед першим рекурсивним викликом функції сортування, а стоп таймера – безпосередньо після нього.

2. Отриманий час роботи алгоритмів переводити у секунди (тобто ділити на CLOCKS_PER_SEC) не потрібно.

ЩЕ БІЛЬША УВАГА!

Якщо в курсовій роботі зазначені вище важливі вимоги до виміру часу роботи алгоритмів будуть проігноровані і, незважаючи на цю вказівку, старт і стоп таймера буде виконано не так, як вказано в цих вимогах та прикладі, то максимально можлива оцінка за курсову роботу відразу зменшується з «A – відмінно» до «B – дуже добре», тобто від оцінки роботи відразу віднімається 6 (шість) штрафних балів. За переведення часу виміру у секунди (ділення на CLOCKS_PER_SEC) додатково знімається ще 3 (три) штрафних бали.

Врахування похибки вимірювання

Оскільки операційна система Windows є багатозадачною і у ній, окрім прикладної програми, одночасно працює багато системних процесів, то отриманий час має похибку.

Для врахування цієї похибки рекомендується використовувати методику усереднення значень вимірів, програма реалізації якої наведена нижче. Сама методика викладена у коментарях цієї програми.

“Common_Vector.h”

```
#ifndef __CommonVector_H__
#define __CommonVector_H__

#define VectorLength 10000

int Vector[VectorLength];

#endif
```

“Measurement.h”

```
#ifndef __Measurement_H__
#define __Measurement_H__

#include <time.h>
*
// Загальна кількість вимірів часу роботи алгоритма
#define measurements_number 28

// Кількість відкинутих початкових вимірів
#define rejected_number 2
```

```
// Кількість відкинутих вимірів з мінімільними значеннями.
// Вона ж дорівнює кількості відкинутих вимірів
// з максимальними значеннями.
#define min_max_number 3

// Масив значень часу роботи алгоритма
extern clock_t  Res[measurements_number];

// Функція обробки і усереднення значень вимірів
// часу роботи алгоритма.
// Повертає усереднене значення часу роботи алгоритма.
float MeasurementProcessing();

#endif
```

“Measurement.c”

```
#include "Measurement.h"
#include <stdio.h>

// Масив значень часу роботи алгоритма
clock_t  Res[measurements_number];

float MeasurementProcessing()
{
   long int Sum;
   float AverageValue;

//////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
   printf("Initial Measurement Results:\n");
   for (int i = 0; i< measurements_number; i++)
      printf("   %ld",Res[i]);
   printf("\n\n");
// У курсовій роботі ці оператори виводу потрібно видалити
//////////////////////////////////////////////////////////
```

```
//-------------------------------------------------------------
```
/* Методика усереднення результатів виміру:
Крок 1. Відкидається **rejected_number** перших вимірів з індексами від 0 до **rejected_number-1**.
Крок 2. Серед інших елементів знаходимо **min_max_number** вимірів з мінімальними значеннями та min_max_number вимірів з максимальними значеннями, і також відкидаємо їх.
Крок 3. Знаходимо середнє значення вимірів, що залишилися.
*/
```
//-------------------------------------------------------------
```

/* Крок 1. Для виконання Кроку 1 просто починаємо алгоритм з індексу **rejected_number**.
*/
/* Крок 2.
Для знаходження **min_max_number** мінімальних і максимальних значень вимірів виконуємо **min_max_number** ітерацій головного цикла алгоритму шейкерного сортування в діапазоні індексів від **rejected_number** до **measurements_number-1**.
В результаті, **min_max_number** мінімальних значень вимірів будуть знаходитись на позиціях в діапазоні індексів від **rejected_number** до **rejected_number+min_max_number-1**, а **min_max_number** максимальних значень вимірів – на позиціях в діапазоні індексів від **measurements_number-min_max_number** до **measurements_number-1**.
*/

```
   clock_t  buf;
     int L = rejected_number, R = measurements_number - 1;
     int k = rejected_number;
     for (int j=0; j < min_max_number; j++) {
          for (int i = L; i < R; i++)    {
               if (Res[i] > Res[i + 1]) {
                    buf = Res[i];
                    Res[i] = Res[i + 1];
                    Res[i + 1] = buf;
                    k = i;
               }
          }
```

```
            R = k;
            for (int i = R - 1; i >= L; i--) {
                  if (Res[i] > Res[i + 1]) {
                        buf = Res[i];
                        Res[i] = Res[i + 1];
                        Res[i + 1] = buf;
                        k = i;
                  }
            }
            L = k + 1;
      }
////////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
   printf("Measurement Results after sorting:\n");
   for (int i = 0; i< measurements_number; i++)
      printf("   %ld",Res[i]);
   printf("\n\n");
// У курсовій роботі ці оператори виводу потрібно видалити
////////////////////////////////////////////////////////////

// Крок 3.
/* Знаходимо середнє значення вимірів після відкидання
rejected_number перших вимірів та min_max_number вимірів з
мінімальними значеннями і min_max_number вимірів з максима-
льними значеннями, тобто середнє значення вимірів в діапазоні
індексів від rejected_number + min_max_number до
measurements_number - min_max_number - 1.
*/
   Sum=0;
   for (int i = rejected_number + min_max_number; i <
measurements_number - min_max_number; i++)
      Sum = Sum + Res[i];

/* Кількість вимірів, що залишилась для обчислення середнього
значення, дорівнює
measurements_number – 2 * min_max_number - rejected_number
*/
```

```
   AverageValue = (float)Sum / (float)(measurements_number - 2*min_max_number - rejected_number);

//////////////////////////////////////////////////////////////
// Завдяки цим операторам виводу ви можете зрозуміти навіщо
// потрібне таке обчислення результатів виміру
   printf("Initial rejected values: ");
   for (int i = 0; i < rejected_number; i++)
      printf("%ld   ", Res[i]);
   printf("\n");

   printf("Rejected minimal values: ");
   for (int i = rejected_number; i < rejected_number + min_max_number; i++)
      printf("%ld   ", Res[i]);
   printf("\n");

   printf("Rejected maximal values: ");
   for (int i = measurements_number - min_max_number; i < measurements_number; i++)
      printf("%ld   ", Res[i]);
   printf("\n\n\n");

   printf("Sum of the rest values, Sum = %ld;\n", Sum);
   printf("AverageValue = %11.2f\n\n\n", AverageValue);
// У курсовій роботі ці оператори виводу потрібно видалити
//////////////////////////////////////////////////////////////

   return AverageValue;

}
```

“main.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "CommonVector.h"
#include "Measurement.h"

void FillVector (int *A, int N)    {
//    for(int i=0; i<N; i++) A[i] = i;

    for(int i=0; i<N; i++) A[i] = rand();

//    for(int i=0; i<N; i++) A[i] = N-i;
//    for(int i=0; i<N; i++) printf(" %d",A[i]);
}

clock_t ExchangeSort(int *A, int N)
{
    int R, i, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(R = N-1; R > 0; R--) {
        for( i = 0 ; i < R; i++)
            if (A[i] > A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
            }
    }

    time_stop = clock();
    return time_stop - time_start;
}
```

```
void ExchangeSortMeasurement()
{
   for (int i = 0; i < measurements_number; i++) {
      FillVector(Vector, VectorLength);
      Res[i] = ExchangeSort(Vector, VectorLength);
   }
}

void OutTable(float ordered, float random, float backordered)
{
// Усереднений результат кожного з трьох вимірів буде виведено
// на екран у портібні позиції
   printf("\t\t Ordered \t Random \t BackOrdered \n");

// В даному прикладі в усіх трьох колонках таблиці вимірів
// друкується одне й те ж саме значення
   printf("Exchange\t %7.2f \t %7.2f \t %7.2f \n", ordered, random,
backordered);
   printf("\n\n");
}

int main()
{
   float SortingTime;

   srand(time(NULL));
   ExchangeSortMeasurement();
   SortingTime = MeasurementProcessing();
// В даному прикладі в усіх трьох колонках таблиці вимірів
// друкується одне й те ж саме значення
   OutTable(SortingTime, SortingTime, SortingTime);

   return 0;
}
```

ЗАДАЧІ

1. Впорядкувати окремо кожен переріз тривимірного масива Arr3D [P,M,N] наскрізно по стовпчиках за незменшенням.
2. Впорядкувати окремо кожен переріз тривимірного масива Arr3D [P,M,N] таким чином: переставити стовпчики перерізу за незменшенням значень елементів його першого рядка.
3. Впорядкувати окремо кожен переріз тривимірного масива Arr3D [P,M,N] таким чином: переставити стовпчики перерізу за незменшенням сум їх елементів.
4. Впорядкувати тривимірний масив Arr3D [P,M,N] таким чином: переставити перерізи масива за незменшенням значень вектору перших елементів кожного перерізу A [*,1,1].
5. Впорядкувати тривимірний масив Arr3D [P,M,N] таким чином: переставити перерізи масива за незменшенням сум їх елементів.

Зауваження:

1) Першим елементом тривимірного масива є Arr3D [1,1,1], останнім – Arr3D [P,M,N]. Зміна третього індексу відбувається швидше за інші, а першого – повільніше, ніж інших індексів.
2) Перерізи тривимірного масива Arr3D [P,M,N] треба брати тільки по першому виміру (індексу). Тобто у масиві Arr3D [P,M,N] треба брати P перерізів.

АЛГОРИТМИ СОРТУВАННЯ

1. Алгоритм сортування №1 методу прямої вставки з лінійним пошуком місця вставки від початку послідовності, що сортується, або «зліва» (додаток 1, рис. №1).

2. Алгоритм сортування №2 методу прямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», без бар'єру (додаток 1, рис. №2).

3. Алгоритм сортування №3 методу прямої вставки з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар'єром (додаток 1, рис. №3).

4. Алгоритм сортування №4 методу прямої вставки з двійковим пошуком місця вставки (додаток 1, рис. №4).

5. Алгоритм сортування №1 методу прямого вибору (додаток 1, рис. №5).

6. Алгоритм сортування №2 методу прямого вибору (додаток 1, рис. №6).

7. Алгоритм сортування №3 методу прямого вибору (додаток 1, рис. №7).

8. Алгоритм сортування №4 методу прямого вибору (додаток 1, рис. №8).

9. Алгоритм сортування №5 методу прямого вибору (додаток 1, рис. №9).

10. Алгоритм сортування №6 методу прямого вибору (додаток 1, рис. №10).

11. Алгоритм сортування №7 методу прямого вибору (додаток 1, рис. №11).

12. Алгоритм сортування №8 методу прямого вибору (додаток 1, рис. №12).

13. Алгоритм сортування №1 методу прямого обміну без модифікацій (додаток 1, рис. №13).

14. Алгоритм сортування №2 методу прямого обміну з використанням прапорця (додаток 1, рис. №14).

15. Алгоритм сортування №3 методу прямого обміну із запам'ятовуванням місця останньої перестановки (додаток 1, рис. №15).

16. Алгоритм сортування №4 методу прямого обміну (Шейкерне сортування) (додаток 1, рис. №16).

17. Гібридний алгоритм "вставка – обмін" (додаток 1, рис. №17).

18. Гібридний алгоритм "вибір№1 – обмін" (додаток 1, рис. №18).

19. Гібридний алгоритм "вибір№2 – обмін" (додаток 1, рис. №19).

20. Гібридний алгоритм "вибір№3 – обмін" (додаток 1, рис. №20).

21. Гібридний алгоритм "вибір№4 – обмін" (додаток 1, рис. №21).

22. Алгоритм №1 методу сортування Шелла (класичний варіант на основі прямої вставки №2) (додаток 1, рис. №22). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.

23. Алгоритм №2 методу сортування Шелла (спосіб реалізації на основі гібридного алгоритму «вставка-обмін») (додаток 1, рис.

№23). Кількість етапів та кроки між елементами на кожному етапі взяти в залежності від довжини послідовностей, що сортуються.

24. Алгоритм методу «швидкого сортування» (сортування Хоара) (додаток 1, рис. №24).

СПОСОБИ ОБХОДУ (ВЗЯТТЯ ЕЛЕМЕНТІВ) МАСИВА ДЛЯ ВИКОНАННЯ СОРТУВАННЯ

1. Переписати елементи заданого двовимірного масива у додатковий одновимірний масив. Виконати сортування. Повернути результат у початковий масив.
2. Не використовуючи додаткового масива, виконати сортування перетворюючи один індекс ***k*** елементів "уявного" вектора у відповідні індекси ***i, j*** елементів конкретного перерізу заданого тривимірного масива за допомогою формул:

   i = k % M;

   j = k / M;

   де ***M*** – кількість рядків (довжина стовпчика) переріза.
3. Виконати сортування, здійснюючи обхід безпосередньо по елементах заданого двовимірного масива, не використовуючи додаткових масивів і перетворень індексів.
4. Використовуючи елементи першого рядка кожного перерізу як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. *При перестановці стовпчиків потрібно саме копіювати їх елементи, а не ко-*

піювати вказівники на них, використовуючи операції з вказівниками мови С/С++.

5. В якості першого етапу сортування сформувати додатковий вектор Sum, довжина якого дорівнює кількості стовпчиків і значеннями якого є суми елементів відповідних стовпчиків. Використовуючи елементи вектора Sum як ключі сортування, переставляти відповідні стовпчики кожен раз, коли треба переставляти ключі. *При перестановці стовпчиків потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови С/С++.*
6. Використовуючи значення вектору перших елементів кожного перерізу Arr3D[*,1,1] як ключі сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі. *При перестановці перерізів потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови С/С++.*
7. В якості першого етапу сортування сформувати додатковий вектор Sum, довжина якого дорівнює кількості перерізів і значеннями якого є суми елементів відповідних перерізів. Використовуючи елементи вектора Sum як ключі сортування, переставляти відповідні перерізи кожен раз, коли треба переставляти ключі. *При перестановці перерізів потрібно саме копіювати їх елементи, а не копіювати вказівники на них, використовуючи операції з вказівниками мови С/С++.*

ВИПАДКИ ВІДСОРТОВАНОСТІ МАСИВА

1. Елементи початкового масива є прямо впорядкованими різними числами згідно умові задачі (тобто **строго** за збільшенням).

```
int number=0;
for (int k=0; k<P; k++)
    for (int j=0; j<N; j++)
            for (int i=0; i<M; i++)
                Arr3D[k][i][j] = number++;
```

2. Елементи початкового масива є випадковими числами з того ж діапазону значень, що й впорядкований та обернено-впорядкований масиви.

```
for (int k=0; k<P; k++)
    for (int j=0; j<N; j++)
        for (int i=0; i<M; i++)
            Arr3D[k][i][j] = rand() % (P*M*N);
```

3. Елементи початкового масива є обернено впорядкованими різними числами згідно умові задачі (тобто **строго** за зменшенням).

```
int number = P*M*N;
for (int k=0; k<P; k++)
    for (int j=0; j<N; j++)
            for (int i=0; i<M; i++)
                Arr3D[k][i][j] = number--;
```

ВИПАДКИ ДОСЛІДЖЕННЯ

Для виконання вимірів і подальшого порівняльного аналізу рекомендується взяти перелічені нижче варіанти розмірів тривимірного масива. Як виключення, тільки у випадку, якщо на комп'ютері студента не вистачає розміру оперативної пам'яті, дозволяється взяти для дослідження масиви з меншими (але максимально можливими для цього комп'ютера) варіантами розміру масива, зберігаючи пропорції між кількістю перерізів, рядків та стовпчиків і загальну логіку випадків порівняння у дослідженні.

Доцільні випадки дослідження для різних задач будуть різними і перелічені нижче.

Для того, щоб виміри були найбільш коректними, розміри тривимірного масива потрібно брати максимально можливими. Для цього, тривимірний масив повинен бути оголошений як динамічний масив, а також повинна бути виділена для нього пам'ять передпочатком роботи і звільнена в кінці роботи програми наступним чином:

```
#include <stdio.h>
#include <stdlib.h>

#define P 3      // Кількість перерізів
#define M 10     // Кількість рядків
#define N 10     // Кількість стовпчиків

int main()
{
// Оголошення динамічного тривимірного масиву
// та виділення для нього пам'яті
```

```
    int ***Arr3D;
    Arr3D = (int***) malloc(P*sizeof(int**));
    for (int k=0; k<P; k++)
    {   Arr3D[k] = (int**) malloc(M*sizeof(int*));
        for (int i=0; i<M; i++)
            Arr3D[k][i] = (int*) malloc(N*sizeof(int));
    }

// Далі до масиву можна звертатись звичайним способом за
// допомогою трьох індексів, наприклад A[k][i][j]

// Введення елементів масиву, впорядкований по стовпчиках за
// збільшенням може виглядати, наприклад, так
    int number=0;
    for (int k=0; k<P; k++)
        for (int j=0; j<N; j++)
                for (int i=0; i<M; i++)
              Arr3D[k][i][j] = number++;

// Після того, як робота з масивом закінчена і він став не потрібним,
необхідно звільнити його пам'ять.
// Це робиться так
    for (int k=0; k<P; k++)
    {   for (int i=0; i<M; i++)
            free(Arr3D[k][i]);
        free(Arr3D[k]);
    }
    free(Arr3D);

// Пам'ять звільнено, можна закінчувати програму
    return 0;
}
```

При такому оголошенні і виділенні пам’яті, подальша робота з таким динамічним масивом нічим не відрізняється від роботи із звичайним масивом.

Випадки дослідження для задачі №1

Випадок дослідження I. Залежність часу роботи алгоритмів від форми перерізу масива.

Рекомендовані розміри масива для досліджень:

P = const = 3, M = var, N = var, M*N = const = 100000000.

1) M = 10; N = 10000000;

1) M = 100; N = 1000000;

2) M = 1000; N = 100000;

3) M = 10000; N = 10000;

4) M = 100000; N = 1000;

5) M = 1000000; N = 100;

6) M = 10000000; N = 10;

Вектор довжиною NV = M*N = 100000000.

Для порівняння час сортування вектора довжиною NV = M*N помножити на P.

Випадок дослідження II. Залежність часу роботи алгоритмів від кількості перерізів масива

Рекомендовані розміри масива для досліджень:

P = var, M = const = 2000, N = const = 2000. (кількість елементів у перерізі M*N = 4000000)

1) P = 2;

2) P = 4;

3) P = 8;

4) P = 16;

5) P = 32;

6) P = 64;

Час сортування вектора довжиною NV = 4000000 (дорівнює кількості елементів у перерізі) помножити на P. Вимір і порівняння для вектора достатньо зробити тільки для найбільшого P.

Випадок дослідження III. Залежність часу роботи алгоритмів від розміру перерізів масива

Порівняння з вектором для цього випадку не обов'язкове.

Для бажаючих порівняти з вектором: тривимірний масив кожного розміру P*M*N (де M=N) потрібно окремо порівнювати з вектором відповідної довжини M*N, і час сортування такого вектора помножити на кількість перерізів P.

Рекомендовані розміри масива для досліджень:

P = const = 3

1) M = N = 4 (M*N = 16)

2) M = N = 8 (M*N = 64)

3) M = N = 16 (M*N = 256)

4) M = N = 32 (M*N = 1024)

5) M = N = 64 (M*N = 4096)

6) M = N = 128 (M*N = 16384)

7) M = N = 256 (M*N = 65535)

8) M = N = 512 (M*N = 262144)

9) M = N = 1024 (M*N = 4194304)

10) M = N = 2048 (M*N = 1048576)

11) M = N = 4096 (M*N = 16777216)

12) M = N = 8192 (M*N = 67108864)

Задачі №2 та №3

Випадок дослідження I. Залежність часу роботи алгоритмів від довжини стовпчиків масива

Рекомендовані розміри масива для досліджень:

Кількість ключів у перерізі (N) і загальна кількість ключів (N*P) є константами.

P = const = 3, N = const = 155000

1) M = 1;

2) M = 2;

3) M = 4;

4) M = 8;

5) M = 16;

6) M = 32;

7) M = 64;

8) M = 128;

9) M = 256;

10) M = 512;

11) M = 1024;

Вектор довжиною кількості ключів у перерізі N = 155000.

Для порівняння час сортування такого вектора помножити на P.

Випадок дослідження II. Залежність часу роботи алгоритмів від форми перерізів масива

Порівняння з вектором для цього випадку не потрібне.

Рекомендовані розміри масива для досліджень:
Кількість елементів у кожному перерізі (M*N) є константою.
P = const = 3, M = var, N = var, M*N = const = 100000000.
1) M = 10; N = 10000000;
1) M = 100; N = 1000000;
2) M = 1000; N = 100000;
3) M = 10000; N = 10000;
4) M = 100000; N = 1000;
5) M = 1000000; N = 100;
6) M = 10000000; N = 10;

Випадок дослідження III. Залежність часу роботи алгоритмів від кількості ключів у кожному перерізі масива при однаковій загальній кількості ключів у всьому масиві

Порівняння з вектором для цього випадку не потрібне.

Рекомендовані розміри масива для досліджень:
Кількість ключів у перерізі N = var
Загальна кількість ключів у матриці P*N = const = 4194304

Довжина стовпчика M = const = 100

P = var, N = var, M = const = 100, P*N = const = 4194304

1) P = 20; N = 200000;

2) P = 200; N = 20000;

3) P = 2000; N = 2000;

4) P = 20000; N = 200;

5) P = 200000; N = 20;

Задачі №4 та №5

Випадок дослідження I. Залежність часу роботи алгоритмів від розміру перерізів масива

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 1500000

Форма перерізу – однакова (квадрат)

M = var, N = var, M = N

1) M = N = 1 (M*N = 1)

2) M = N = 2 (M*N = 4)

3) M = N = 4 (M*N = 16)

4) M = N = 8 (M*N = 64)

5) M = N = 16 (M*N = 256)

Вектор довжиною P = 1500000

Випадок дослідження ІІ. Залежність часу роботи алгоритмів від форми перерізів масива

Рекомендовані розміри масива для досліджень:

Кількість ключів (перерізів) P = const = 128000

M = var, N = var, M*N = const

Загальна кількість елементів P*M*N = const

1) M = 2; N = 800;

2) M = 4; N = 400;

3) M = 8; N = 200;

4) M = 16; N = 100;

5) M = 100; N = 16;

6) M = 200; N = 8;

7) M = 400; N = 4;

8) M = 800; N = 2;

Вектор довжиною P = 128000

ВАРІАНТИ ІНДИВІДУАЛЬНИХ ЗАВДАНЬ

Початковий номер варіантів для крупи з шифром **КВ-ХУ** вираховується за формулою:

(25*X + 32*(Y – 1) + 1) mod 160

Варіант	Задача	Номери алгоритмів згідно нумерації у розділі «Алгоритми сортування» (в дужках – окремий спосіб обходу)	Способи обходу
1	1	7	1 – 3
2	2	1, 6, 22	4
3	1	22, 23, 24	2
4	3	14, 12, 23	5
5	4	11, 8, 24	6
6	1	1, 13, 7	3
7	5	21, 10, 12	7
8	1	2, 13, 8	3
9	2	2, 8, 23	4
10	3	15, 6, 24	5
11	1	5, 13, 11	3
12	4	9, 10, 22	6
13	1	6, 13, 12	3
14	5	20, 6, 23	7
15	2	3, 10, 24	4
16	1	14 (3), 7 (3), 22 (2)	в дужках
17	3	16, 8, 22	5
18	1	9, 13, 20	3
19	4	7, 12, 23	6
20	5	19, 8, 24	7
21	1	10, 13, 21	3
22	2	4, 12, 22	4
23	1	2, 14, 7	3

24	3	17, 10, 23	5
25	4	5, 6, 24	6
26	1	8	1 – 3
27	5	18, 10, 22	7
28	1	5, 14, 8	3
29	2	5, 22, 23	4
30	3	18, 22, 24	5
31	1	6, 14, 11	3
32	4	4, 23, 24	6
33	1	9, 14, 12	3
34	5	17, 12, 23	7
35	2	7, 6, 23	4
36	1	10, 14, 20	3
37	3	19, 12, 24	5
38	1	1, 14, 21	3
39	4	3, 8, 22	6
40	5	16, 22, 23	7
41	1	15 (3), 8 (3), 23 (2)	в дужках
42	2	9, 8, 24	4
43	1	7, 22, 24	2
44	3	20, 6, 22	5
45	4	2, 10, 23	6
46	1	5, 15, 7	3
47	5	15, 6, 24	7
48	1	6, 15, 8	3
49	2	11, 10, 22	4
50	3	21, 8, 23	5
51	1	9, 15, 11	3
52	4	1, 12, 24	6
53	1	11	1 – 3
54	5	14, 8, 22	7
55	2	14, 12, 23	4
56	1	10, 15, 12	3
57	3	1, 10, 24	5
58	1	1, 15, 20	3

59	4	21, 6, 22	6
60	5	11, 10, 23	7
61	1	2, 15, 21	3
62	2	15, 22, 24	4
63	1	6, 17, 7	3
64	3	2, 22, 23	5
65	4	20, 10, 12	6
66	1	9, 17, 8	3
67	5	9, 12, 24	7
68	1	1 (3), 11 (3), 24 (2)	в дужках
69	2	16, 6, 24	4
70	3	3, 12, 22	5
71	1	10, 17, 11	3
72	4	19, 8, 23	6
73	1	1, 17, 12	3
74	5	5, 22, 24	7
75	2	17, 8, 22	4
76	1	2, 17, 20	3
77	3	4, 6, 23	5
78	1	5, 17, 21	3
79	4	18, 10, 24	6
80	5	7, 6, 22	7
81	1	12	1 – 3
82	2	18, 10, 23	4
83	1	8, 23, 24	2
84	3	5, 8, 24	5
85	4	17, 12, 22	6
86	1	9, 13, 7	3
87	5	4, 8, 23	7
88	1	10, 13, 8	3
89	2	19, 12, 24	4
90	3	7, 10, 22	5
91	1	1, 13, 11	3
92	4	16, 6, 23	6
93	1	2, 13, 12	3

94	5	3, 10, 24	7
95	2	20, 23, 24	4
96	1	2 (3), 12 (3), 22 (2)	в дужках
97	3	9, 23, 24	5
98	1	5, 13, 20	3
99	4	15, 22, 23	6
100	5	2, 12, 22	7
101	1	6, 13, 21	3
102	2	21, 6, 22	4
103	1	10, 14, 7	3
104	3	11, 12, 23	5
105	4	14, 8, 24	6
106	1	20	1 – 3
107	5	1, 23, 24	7
108	1	1, 14, 8	3
109	2	1, 8, 23	4
110	3	16, 6, 24	5
111	1	2, 13, 11	3
112	4	7, 10, 22	6
113	1	5, 14, 12	3
114	5	21, 6, 23	7
115	2	2, 10, 24	4
116	1	6, 14, 20	3
117	3	17, 8, 22	5
118	1	9, 14, 21	3
119	4	5, 12, 23	6
120	5	20, 8, 24	7
121	1	5 (3), 20 (3), 23 (2)	в дужках
122	2	5, 12, 22	4
123	1	21, 23, 24	2
124	3	20, 10, 12	5
125	4	2, 6, 24	6
126	1	1, 15, 7	3
127	5	19, 10, 22	7
128	1	2, 15, 8	3

129	2	7, 10, 12	4
130	3	21, 10, 23	5
131	1	5, 15, 11	3
132	4	1, 22, 24	6
133	1	21	1 – 3
134	5	18, 12, 23	7
135	2	14, 6, 23	4
136	1	6, 15, 12	3
137	3	3, 12, 24	5
138	1	9, 15, 20	3
139	4	19, 8, 22	6
140	5	17, 6, 24	7
141	1	10, 15, 21	3
142	2	15, 8, 24	4
143	1	2, 17, 7	3
144	3	4, 6, 22	5
145	4	18, 10, 23	6
146	1	5, 17, 8	3
147	5	16, 8, 22	7
148	1	6 (3), 21 (3), 24 (2)	в дужках
149	2	18, 10, 22	4
150	3	9, 8, 23	5
151	1	6, 17, 11	3
152	4	15, 12, 24	6
153	1	9, 17, 12	3
154	5	14, 10, 23	7
155	2	21, 12, 23	4
156	1	10, 17, 20	3
157	3	11, 10, 24	5
158	1	1, 17, 21	3
159	4	14, 6, 22	6
160	5	15, 12, 24	7

СПИСОК РЕКОМЕНДОВАНОЇ ЛІТЕРАТУРИ

1. Вирт Н. Алгоритмы и структуры данных. М.: Мир, 1989.

2. Т.Кормен, Ч.Лейзерзон, Р.Ривест. Алгоритмы: построение и анализ. М.: МЦНМО, 2000.

3. Ахо А., Хопкрофт Дж., Ульман Дж. Структуры данных и алгоритмы., М.: Издательский дом «Вильямс», 2003. – 384 с.

4. Седжвик Р. Алгоритмы на С++., 2-е изд., М.: Интуит, 2016. – 1000 с.

5. Кнут Д. Искусство программирования для ЭВМ. т.1. Основные алгоритмы., М.: Мир, 1976.

6. Кнут Д. Искусство программирования для ЭВМ. т.2. Получисленные алгоритмы., М.: Мир, 1977.

7. Кнут Д. Искусство программирования для ЭВМ. т.3. Сортировка и поиск., М.: Мир, 1978.

8. Марченко А.И., Марченко Л.А. Программирование в среде Turbo Pascal 7.0. – 9-е изд. – К.: Век+, Спб.: КОРОНА-Век, 2007. – 464 с., ил.

9. Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов., М.: Мир, 1979.

10. Гудман С., Хидетниеми С. Введение в разработку и анализ алгоритмов., М.: Мир, 1981.

11. Дал У., Дейкстра., Хоор К. Структурное программирование., М.: Мир, 1975.

ДОДАТОК 1.
АЛГОРИТМИ ДЛЯ ВИКОНАННЯ КУРСОВОЇ РОБОТИ

Для всіх нижченаведених алгоритмів, крім двох, оголошення одномірного масива (вектора) чисел та виклик функцій даних алгоритмів виконується так, як в прикладі виміру часу в розділі «Інструкції з виміру часу роботи алгоритмів».

Виключеннями є «Алгоритм сортування №3 методу прямої вставки (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар'єром)» та «Алгоритм методу «швидкого сортування» (сортування Хоара)».

У алгоритмі прямої вставки №3 використовується «бар'єр», для чого масив оголошується розміром на один елемент більше і сортування виконується в діапазоні індексів від 1 до N. Оголошення вектора для цього алгоритма наведено на рис. 3 зверху від функції алгоритма.

Другим виключенням є «Алгоритм методу «швидкого сортування» (сортування Хоара)», який є рекурсивним алгоритмом. У зв'язку з цим, оскільки функція `QuickSort` є рекурсивною, то виклики функції `clock()` для фіксації часу початку і закінчення сортування, на відміну від інших алгоритмів, як показано на рис.24, винесені з власне функції сортування `QuickSort` у функцію `QuickSortMeasurement`, яка виконує формування масиву і виклик рекурсивної функції `QuickSort`.

УВАГА! Винесення викликів функції `clock()` з власне сортуючої функції у функцію, з якої вона викликається, є коректним тільки для «Алгоритму методу «швидкого сортування» (сортування Хоара)», який є рекурсивним алгоритмом. Для всіх інших 23-ох алгоритмів виклики функції `clock()` повинні бути виконані строго як показано в прикладі виміру часу в розділі «Інструкції з виміру часу роботи алгоритмів» (див. «ВАЖЛИВІ ВИМОГИ ДО ВИМІРУ ЧАСУ» цього розділу).

```
clock_t Insert1(int *A, int N)
{
    int Elem, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        Elem=A[i];
        j=0;
        while (Elem>A[j]) j=j+1;
        for (int k=i-1; k>=j; k--)
            A[k+1]=A[k];
        A[j]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 1. Алгоритм сортування №1 методу прямої **вставки** (з лінійним пошуком місця вставки від початку послідовності, що сортується , або «зліва»)

```
clock_t Insert2(int *A, int N)
{
    int Elem, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        Elem=A[i];
        j=i;
        while (j>0 && Elem<A[j-1]) {
            A[j]=A[j-1];
            j=j-1;
        }
        A[j]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 2. Алгоритм сортування №2 методу прямої **вставки** (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», без бар'єру).

```
#define VectorLength 10

int Vector[VectorLength+1];

-------------------------------------

clock_t Insert3(int *A, int N)
{
    int j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=2; i<N+1; i++){
        A[0]=A[i];
        j=i;
        while (A[0]<A[j-1]) {
            A[j]=A[j-1];
            j=j-1;
        }
        A[j]=A[0];
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 3. Алгоритм сортування №3 методу прямої **вставки** (з лінійним пошуком місця вставки від елемента, що вставляється, або «справа», з бар'єром).

```
clock_t Insert4(int *A, int N)
{
    int Elem, L, R, j;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<=N-1; i++){
        Elem=A[i];
        L=0;  R=i;
        while (L<R){
            j=(L+R)/2;
            if (A[j]<=Elem){
                L=j+1;
            } else R=j;
        }
        for (int k=i-1; k>=R; k--) {
            A[k+1]=A[k];
        }
        A[R]=Elem;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 4. Алгоритм сортування №4 методу прямої **вставки** (з двійковим пошуком місця вставки).

```
clock_t Select1(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s]; imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<Min){
                Min=A[i];
                imin=i;
            }
        A[imin]=A[s];
        A[s]=Min;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 5. Алгоритм сортування №1 методу прямого **вибору**.

```
clock_t Select2(int *A, int N)
{
    int imin, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<A[imin]) imin=i;
        tmp=A[imin];
        A[imin]=A[s];
      A[s]=tmp;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 6. Алгоритм сортування №2 методу прямого **вибору**.

```
clock_t Select3(int *A, int N)
{
    int Min, Max;
    int L, R, imin, imax;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        Min=A[L]; imin=L;
        Max=A[L]; imax=L;
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                imin=i;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    imax=i;
                }
        }
        A[imin]=A[L];
        A[L]=Min;
        if (imax==L) A[imin]=A[R];
        else         A[imax]=A[R];
        A[R]=Max;

        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 7. Алгоритм сортування №3 методу прямого **вибору**.

```
clock_t Select4(int *A, int N)
{
    int L, R, imin, imax, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        imin=L; imax=L;

        for(int i=L+1; i<R+1; i++)
            if (A[i]<A[imin]) imin=i;
            else
                if (A[i]>A[imax]) imax=i;

        tmp=A[imin];
        A[imin]=A[L];
        A[L]=tmp;
        if (imax==L) {
            tmp=A[imin];
            A[imin]=A[R];
            A[R]=tmp;
        }
        else {
            tmp=A[imax];
            A[imax]=A[R];
            A[R]=tmp;
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 8. Алгоритм сортування №4 методу прямого **вибору**.

```
clock_t Select5(int *A, int N)
{
    int Min, imin;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s]; imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<Min){
                Min=A[i];
                imin=i;
            }
        if (imin!=s) {
            A[imin]=A[s];
            A[s]=Min;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 9. Алгоритм сортування №5 методу прямого **вибору**.

```
clock_t Select6(int *A, int N)
{
    int imin, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        imin=s;
        for(int i=s+1; i<N; i++)
         if (A[i]<A[imin]) imin=i;
        if (imin!=s) {
            tmp=A[imin];
            A[imin]=A[s];
            A[s]=tmp;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 10. Алгоритм сортування №6 методу прямого **вибору**.

```
clock_t Select7(int *A, int N)
{
    int Min, Max;
    int L, R, imin, imax;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        Min=A[L]; imin=L;
        Max=A[L]; imax=L;
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                imin=i;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    imax=i;
                }
        }
        if (imin!=L) {
            A[imin]=A[L];
            A[L]=Min;
        }
        if (imax!=R){
            if (imax==L) A[imin]=A[R];
            else         A[imax]=A[R];
            A[R]=Max;
        }
        L=L+1; R=R-1;
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

Рис. 11. Алгоритм сортування №7 методу прямого **вибору**.

```
clock_t Select8(int *A, int N)
{
    int L, R, imin, imax, tmp;

    clock_t time_start, time_stop;
    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        imin=L; imax=L;

        for(int i=L+1; i<R+1; i++)
            if (A[i]<A[imin]) imin=i;
            else
                if (A[i]>A[imax]) imax=i;

        if (imin!=L) {
            tmp=A[imin];
            A[imin]=A[L];
            A[L]=tmp;
        }
        if (imax!=R)
            if (imax==L) {
                tmp=A[imin];
                A[imin]=A[R];
                A[R]=tmp;
            }
            else {
                tmp=A[imax];
                A[imax]=A[R];
                A[R]=tmp;
            }
        L=L+1; R=R-1;
    }
    time_stop = clock();
    return time_stop - time_start;
}
```

Рис. 12. Алгоритм сортування №8 методу прямого **вибору**.

```
clock_t Exchange1(int *A, int N)
{
    int tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int R=N-1; R>0; R--){
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 13. Алгоритм сортування №1 методу прямого **обміну** (без модифікацій).

```
clock_t Exchange2(int *A, int N)
{
    int R, flag, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    R=N-1; flag=1;
    while(flag == 1){
        flag=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                flag=1;
            }
        R--;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 14. Алгоритм сортування №2 методу прямого **обміну** (з використанням прапорця).

```
clock_t Exchange3(int *A, int N)
{
    int R, k, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    R=N-1;
    while(R>0){
        k=0;
        for(int i=0; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        R=k;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 15. Алгоритм сортування №3 методу прямого **обміну** (із запам'ятовуванням місця останньої перестановки).

```
clock_t Exchange4(int *A, int N)
{
    int L, R, k, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1; k=0;
    while(L<R){
        for(int i=L; i<R; i++)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        R=k;
        for(int i=R-1; i>=L; i--)
            if (A[i]>A[i+1]) {
                tmp=A[i];
                A[i]=A[i+1];
                A[i+1]=tmp;
                k=i;
            }
        L=k+1;
     }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 16. Алгоритм сортування №4 методу прямого **обміну** (Шейкерне сортування).

```
clock_t InsertExchange(int *A, int N)
{
    int j, tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int i=1; i<N; i++){
        j=i;
        while (j>0 && A[j]<A[j-1]) {
            tmp=A[j];
            A[j]=A[j-1];
            A[j-1]=tmp;
            j=j-1;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 17. Гібридний алгоритм "вставка – обмін".

```
clock_t Select1Exchange(int *A, int N)
{
    int Min;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        Min=A[s];
        for(int i=s+1; i<N; i++)
            if (A[i]<Min){
                Min=A[i];
                A[i]=A[s];
                A[s]=Min;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 18. Гібридний алгоритм "вибір№1 – обмін".

```
clock_t Select2Exchange(int *A, int N)
{
    int tmp;
    clock_t time_start, time_stop;

    time_start = clock();

    for(int s=0; s<N-1; s++){
        for(int i=s+1; i<N; i++)
            if (A[i]<A[s]){
                tmp=A[i];
                A[i]=A[s];
                A[s]=tmp;
            }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 19. Гібридний алгоритм "вибір№2 – обмін".

```
clock_t Select3Exchange(int *A, int N)
{
    int Min, Max, tmp;
    int L, R;

    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        if (A[L] > A[R]) {
            tmp=A[L];
            A[L]=A[R];
            A[R]=tmp;
        }
        Min=A[L]; Max=A[R];
        for(int i=L+1; i<R+1; i++){
            if (A[i] < Min){
                Min=A[i];
                A[i]=A[L];
                A[L]=Min;
            }
            else
                if (A[i] > Max){
                    Max=A[i];
                    A[i]=A[R];
                    A[R]=Max;
                }
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 20. Гібридний алгоритм "вибір№3 – обмін".

```
clock_t Select4Exchange(int *A, int N)
{
    int Min, Max;
    int L, R;

    clock_t time_start, time_stop;

    time_start = clock();

    L=0; R=N-1;
    while (L<R){
        for(int i=L; i<R+1; i++){
            if (A[i] < A[L]){
                Min=A[i];
                A[i]=A[L];
                A[L]=Min;
            }
            else
                if (A[i] > A[R]){
                    Max=A[i];
                    A[i]=A[R];
                    A[R]=Max;
                }
        }
        L=L+1; R=R-1;
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 21. Гібридний алгоритм "вибір№4 – обмін".

```
clock_t Shell_1(int *A, int N)
{
    int Elem, t, j, k;
    clock_t time_start, time_stop;

    time_start = clock();

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;

    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
        Stages[i]=2*Stages[i+1]+1;

    for (int p=0; p<t; p++){
        k=Stages[p];
        for (int i=k; i<N; i++){
            Elem=A[i];
            j=i;
            while (j>=k && Elem<A[j-k]) {
                A[j]=A[j-k];
                j=j-k;
            }
            A[j]=Elem;
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 22. Алгоритм №1 методу сортування Шелла.

```
clock_t Shell_2(int *A, int N)
{
    int tmp, t, j, k;
    clock_t time_start, time_stop;

    time_start = clock();

    if (N<4) t=1;
    else t=(int)log2f((float)N)-1;

    Stages[t-1]=1;
    for (int i=t-2; i>=0; i--)
      Stages[i]=2*Stages[i+1]+1;

    for (int p=0; p<t; p++){
        k=Stages[p];
        for (int i=k; i<N; i++){
            j=i;
            while (j>=k && A[j]<A[j-k]) {
                tmp=A[j];
                A[j]=A[j-k];
                A[j-k]=tmp;
                j=j-k;
            }
        }
    }

    time_stop = clock();

    return time_stop - time_start;
}
```

Рис. 23. Алгоритм №2 методу сортування Шелла.

```
void QuickSort(int L, int R)
{
    int B, tmp, i, j;

    B=Vector[(L+R)/2];
    i=L; j=R;
    while (i<=j) {
        while (Vector[i] < B) i=i+1;
        while (Vector[j] > B) j=j-1;
        if (i<=j) {
            tmp=Vector[i];
            Vector[i]=Vector[j];
            Vector[j]=tmp;
            i=i+1;
            j=j-1;
        }
    }
    if (L<j) QuickSort(L,j);
    if (i<R) QuickSort(i,R);
}

void QuickSortMeasurement()
{
    clock_t time_start, time_stop;

    for (int i=0; i < measurements_number; i++)
    {
        FillVector(Vector, VectorLength);
        time_start = clock();
        QuickSort(0, VectorLength-1);
        time_stop = clock();
        Res[i] = time_stop - time_start;
    }
}
```

Рис. 24. Алгоритм методу «швидкого сортування» (сортування Хоара).

ДОДАТОК 2. ПЕРШИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

КУРСОВА РОБОТА

з дисципліни "Структури даних і алгоритми"

Виконав: Прізвище Ініціали

Група: КВ-??

Номер залікової книжки: КВ-????

Допущений до захисту

_____

2 семестр 20__/20__ навч.року

ДОДАТОК 3. ДРУГИЙ ТИТУЛЬНИЙ ЛИСТ КУРСОВОЇ РОБОТИ.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ «КПІ ім. Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Узгоджено	ЗАХИЩЕНА "__"_____20__р.
Керівник роботи	з оцінкою_____
_____/Марченко О.І./	_____ /Марченко О.І./

Дослідження ефективності методів сортування (назви конкретних методів сортування) на багатовимірних масивах

Виконавець роботи:_____

(підпис)

Прізвище Ім'я По батькові

_____20__ р.

ДОДАТОК 4. СТРУКТУРА ТЕХНІЧНОГО ЗАВДАННЯ КУРСОВОЇ РОБОТИ.

ТЕХНІЧНЕ ЗАВДАННЯ НА КУРСОВУ РОБОТУ

I. Описати теоретичні положення, від яких відштовхується дослідження, тобто принцип та схему роботи кожного із досліджуваних алгоритмів сортування для одновимірного масива, навести загальновідомі властивості цих алгоритмів та оцінки кількості операцій порівняння та присвоєння для них.

II. Скласти алгоритми рішення задачі сортування в багатовимірному масиві заданими за варіантом методами та написати на мові програмування за цими алгоритмами програму, яка відповідає вимогам розділу «Вимоги до програми курсової роботи».

III. Виконати налагодження та тестування коректності роботи написаної програми.

IV. Провести практичні дослідження швидкодії складених алгоритмів, тобто виміри часу роботи цих алгоритмів для різних випадків та геометричних розмірів багатовимірних масивів.

V. За результатами досліджень скласти порівняльні таблиці за різними ознаками.

- Одна таблиця результатів (вимірів часу сортування впорядкованого, випадкового і обернено-впорядкованого масива) для масива з заданими геометричними розмірами повинна бути такою:

Таблиця № для масива A[P,M,N], де P= ; M= ; N= ;

	Впорядкований	Невпорядкований	Обернено впорядкований
Назва алгоритму 1			
Назва алгоритму 2			
Назва алгоритму 3			

УВАГА! Колонки таблиць виміру часу в програмі та у звіті курсової роботи місцями НЕ переставляти! За невиконання цієї вимоги нараховується 3 (три) штрафних бали.

- Для варіантів курсової роботи, де крім алгоритмів порівнюються також способи обходу, в назвах рядків таблиць потрібно вказати як назви алгоритмів, так і номери способів обходу.
- Для виконання ґрунтовного аналізу алгоритмів потрібно зробити виміри часу та побудувати таблиці для декількох масивів з різними геометричними розмірами.
- Зробити виміри часу для стандартного випадку одномірного масива, довжина якого вибирається такою, щоб можна було виконати коректний порівняльний аналіз з рішенням цієї ж задачі для багатовимірного масива.
- Кількість необхідних таблиць для масивів з різними геометричними розмірами залежить від задачі конкретного варіанту курсової роботи і вибираються так, щоб виконати всебічний та ґрунтовний порівняльний аналіз заданих алгоритмів.
- Рекомендації випадків дослідження з різними геометричними розмірами масивів наведені у розділі «Випадки дослідження».

VI. Для наочності подання інформації за отриманими результатами рекомендується також будувати стовпчикові діаграми та графіки.

VII. Виконати порівняльний аналіз поведінки заданих алгоритмів за отриманими результатами (вимірами часу):

- для одномірного масива відносно загальновідомої теорії;
- для багатовимірних масивів відносно результатів для одномірного масива;
- для заданих алгоритмів на багатовимірних масивах між собою;
- дослідити вплив різних геометричних розмірів багатовимірних масивів на поведінку алгоритмів та їх взаємовідношення між собою;
- **для всіх вищезазначених пунктів порівняльного аналізу пояснити, ЧОМУ алгоритми в розглянутих ситуаціях поводять себе саме так, а не інакше**.

УВАГА! Саме якість виконання порівняльного аналізу дозволяє отримати оцінки від B (85 балів) до A (100 балів). Якщо в звіті в якості порівняльного аналізу наведений просто словесний опис фактів, що й так видно з таблиць, діаграм та графіків, то оцінка курсової роботи буде не вище, ніж C (75 балів).

VIII. Зробити висновки за зробленим порівняльним аналізом.

IX. Програму курсової роботи під час її захисту ОБОВ'ЯЗКОВО мати при собі на електронному носії інформації.

Варіант № ___

Задача

Умова задачі за варіантом.

Досліджувані методи та алгоритми

Перелік методів та алгоритмів за варіантом.

Способи обходу

Перелік способів обходу за варіантом.

Випадки дослідження

Перелік випадків дослідження, які потрібно розглянути для заданої за варіантом задачі згідно розділу «Випадки дослідження».